Race Condition Leads to Persistent Data Corruption and Service Collapse

High **0xJacky** published **GHSA-m468-xcm6-fxg4** 3 days ago

Package

/0xJacky/nginx-ui (Go)

Affected versions	Patched versions
< = 2.3.3	2.3.4

github.com/uozi-tech/cosy (Go)

< 1.30.0	1.30.1

Description

Summary

The `nginx-ui` application is vulnerable to a **Race Condition**. Due to the complete absence of synchronization mechanisms (Mutex) and non-atomic file writes, concurrent requests lead to the severe corruption of the primary configuration file (`app.ini`). This vulnerability results in a persistent Denial of Service (DoS) and introduces a non-deterministic path for **Remote Code Execution (RCE)** through configuration cross-contamination.

Details

The vulnerability exists because the settings update pipeline does not implement any synchronization primitives. When multiple requests reach the handler simultaneously:

1. **Memory Corruption**: `ProtectedFill()` modifies shared global singleton pointers without thread-safety, leading to inconsistent states in memory.
2. **File Corruption**: The underlying library (`gopkg.in/ini.v1`) performs direct overwrites. Concurrent write operations interleave at the OS level, resulting in `app.ini` files with empty leading lines, truncated fields, or partially overwritten configuration keys.
3. **State Persistent Failure**: Depending on which bytes are corrupted, the application either fails its "is-installed" check (redirecting to `/install`) or encounters a fatal error during boot/runtime that

prevents the process from responding to any further requests.

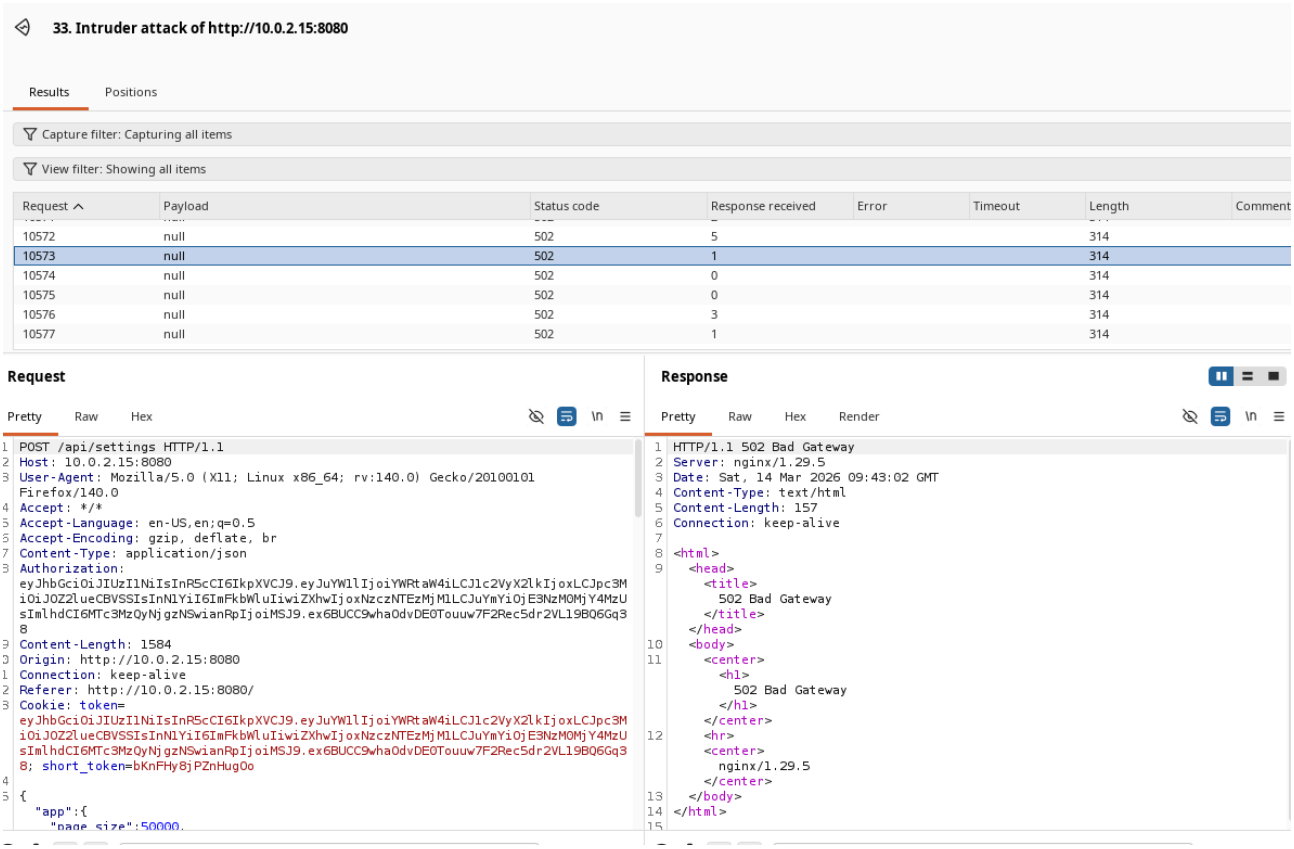
Environment:

- **OS**: Kali Linux 6.17.10-1kali1 (6.17.10+kali-amd64)
- **Application Version**: nginx-ui v2.3.3 (513) `e5da6dd` (go1.26.0)
- **Deployment**: Docker Container

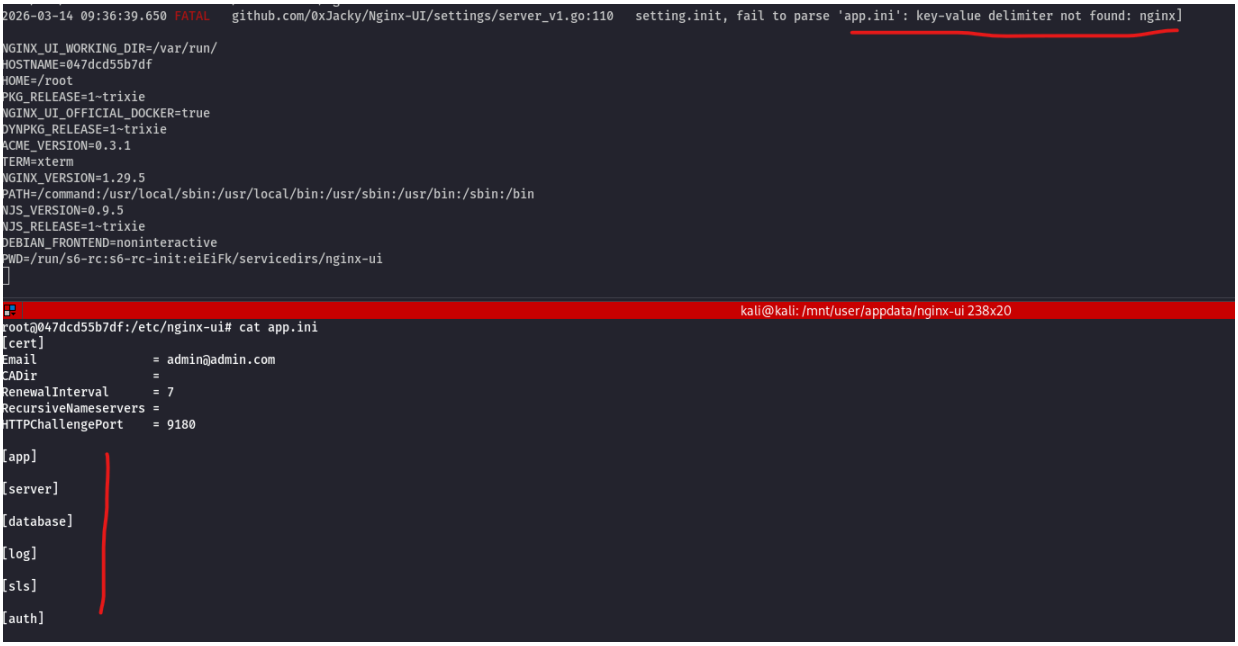
PoC

0. Check original app.ini file valid state:

1. Log in to the `nginx-ui` dashboard.
2. Navigate to Preferences and update settings. Capture a `POST /api/settings` request and send it to **Burp Suite Intruder**.
3. Configure the attack with **Null payloads** (to test basic concurrency) or a **Fuzzing list** (to test data-driven corruption).
4. Set the **Resource Pool** to 20-50 concurrent requests.



5. **Observation (In-flight corruption)**: Monitor the `app.ini` file. You will observe the file being written with empty leading lines or incomplete key-value pairs.



6. **Observation (Recovery Failure)**: If the service redirects to `/install`, attempting to complete the setup again often fails because the underlying configuration state is too corrupted to be reconciled by the installer logic.
7. **Observation (Total Service Collapse)**: When the corruption in `app.ini` becomes so severe, the Go runtime or the INI parser encounters a fatal error, causing the Nginx-UI service to stop responding entirely (Hard DoS).

8. **Observation (Cross-Section Contamination)**: During testing, it was observed that sometimes INI sections become interleaved. For example, fields belonging to the `[nginx]` section (like `ConfigDir` or `ReloadCmd`) were erroneously written under the `[webauthn]` section.

   Example of corrupted output observed:

```
[webauthn]
RPDisplayName  =
RPID           =
RPOrigins      =
```

```
gDirWhiteList  =
ConfigDir      = /etc/nginx
ConfigPath     =
PIDPath        = /run/nginx.pid
SbinPath       =
TestConfigCmd  =
ReloadCmd      = nginx -s reload
RestartCmd     = nginx -s stop
StubStatusPort = 51820
ContainerName  =
```

Impact

This is a **High** security risk (CWE-362: Race Condition).

- **Integrity**: Permanent corruption of application settings and system-level configuration.
- **Availability**: High. The attack results in a persistent Denial of Service that cannot be recovered via the web UI.
- **Remote Code Execution (RCE)** Risk: Since the application allows updating certain fields (like Node Name) and uses others as shell commands (like ReloadCmd or RestartCmd), the observed "cross-contamination" of INI values means an attacker could potentially force a user-controlled string into a command execution field. If ReloadCmd is overwritten with a malicious payload provided in another field, the next nginx reload will execute that payload. While highly impactful, this specific exploit path is non-deterministic and depends on the precise interleaving of thread execution, making targeted exploitation difficult.

Recommended Mitigation

1. **Implement Mutex Locking**: Wrap the `ProtectedFill` and `settings.Save()` calls in a `sync.Mutex` to serialize access to global settings.
2. **Atomic File Writes**: Implement a "write-then-rename" strategy. Write the new configuration to `app.ini.tmp` and use `os.Rename()` to replace the original file atomically, ensuring the configuration file is always in a valid state.

Severity

High 7.1 / 10

CVSS v4 base metrics

Exploitability Metrics

Attack Vector	Network
Attack Complexity	Low
Attack Requirements	None

Privileges Required	High
User interaction	None
Vulnerable System Impact Metrics	
Confidentiality	Low
Integrity	High
Availability	High
Subsequent System Impact Metrics	
Confidentiality	None
Integrity	None
Availability	None

Learn more about base metrics

CVSS:4.0/AV:N/AC:L/AT:N/PR:H/UI:N/VC:L/VI:H/VA:H/SC:N/SI:N/SA:N

CVE ID

CVE-2026-33028

Weaknesses

▶ CWE-362

Credits

dapickle Reporter